UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Solução integrada de suporte ao IT alavancado no paradigma do New IT

André Filipe Neves Vieira

Mestrado em Engenharia Informática
Especialização em Sistemas de Informação

Versão Pública

Trabalho de Projeto orientado por:
Prof. Doutor Luís Manuel Ferreira Fernandes Moniz
Engenheiro Helder Alexandre Moniz

2018

Agradecimentos

Uma vez que este trabalho marca o final de mais uma importante etapa da minha vida, quero aqui aproveitar para deixar um enorme agradecimento a todos aqueles que contribuíram direta ou indiretamente para a concretização desta.

Quero agradecer à empresa Accenture pela oportunidade de poder fazer parte de um projeto desafiante e que contribuiu fortemente para o meu desenvolvimento não só profissional como também pessoal. E um obrigado também à equipa do projeto onde fui integrado por toda a disponibilidade, paciência e apoio técnico dado durante a realização deste trabalho.

Aos meus orientadores, o professor Luís Moniz e Helder Moniz e aos meus colegas Bruno Conceição e Rui Lageira, um obrigado por todo o criticismo construtivo e total disponibilidade para apoio à realização deste trabalho.

A todos os meus amigos, tanto os do Entroncamento como os de Lisboa, um enorme obrigado pelo forte apoio em momentos mais difíceis desta etapa e pelos excelentes momentos que tive o privilégio de partilhar com cada um de vós ao longo de toda a minha vida pessoal e do meu percurso académico.

Por último, um eterno obrigado ao meus pais, Ana Paula e Amílcar Manuel, e ao meu irmão Pedro por me ajudarem a ser aquilo que sou hoje e por tudo o apoio, carinho e confiança que me transmitiram ao longo de toda a minha vida e por tornarem possível a conclusão desta etapa.

À minha família e amigos.

Resumo

Nos últimos anos tem existido uma crescente competitividade entre as empresas de desenvolvimento de software, o que leva também a uma crescente preocupação da agilização e automatização de várias tarefas de desenvolvimento dos seus projetos, dando uma maior garantia de qualidade e eficiência no desenvolvimento dos mesmos. São várias as metodologias e técnicas utilizadas para atingir esse objetivo, sendo as mais conhecidas e mais postas em prática, a Agile e a DevOps. Como tal, e seguindo esta tendência, a Accenture tem procurado implementar metodologias de desenvolvimento mais ágeis nos seus projetos tornando-se assim mais competitiva face a outras empresas no mercado. A esta nova tendência de inovar e agilizar os projetos é, por eles, dado o nome de *New IT*.

O projeto onde este trabalho foi inserido é responsável pelo desenvolvimento aplicacional e manutenção de um sistema de comercialização de energia para uma empresa comercializadora neste mercado. Neste é, diariamente, realizado todo o processo de negócio desde o registo de novos clientes até à faturação dos mesmos. Como tal, a qualidade e rigor no desenvolvimento deste projeto é uma das principais prioridades da empresa e das equipas envolvidas.

Uma vez que, no âmbito do *New IT*, é encorajada a automatização de tarefas, seguindo a metodologia *DevOps* e não existindo ainda muito trabalho desenvolvido neste âmbito, no projeto, este trabalho incidiu, principalmente, sobre a análise e implementação de testes automáticos.

Ao longo deste relatório será apresentado e descrito, com detalhe, todo o trabalho realizado neste âmbito desde a análise, recuperação e operacionalização de testes automáticos existentes, análise e implementação de novos casos de teste, integração destes numa plataforma de desenvolvimento contínua e de modo a que fosse permitido informar as equipas sobre eventuais falhas dos testes. Para além disto foi também realizada uma análise de possíveis formas de avaliar a cobertura dos testes sobre o código do projeto fornecendo assim algumas métricas essenciais para a compreensão e análise da qualidade e rigor com que o código da aplicação é testado.

Palavras-chave: *New IT*, DevOps, Testes automáticos, Cobertura, Automatização, Integração Contínua

Abstract

In recent years there has been a growing competition between most of the software development companies. This competition leads to a growing concern about the agility and the ability to effectively automate the various tasks related to software development. This is done in order to give both project teams and companies a greater guarantee of the quality and efficiency of their development pipelines and methodologies.

There are currently several methodologies and techniques used in order to achieve this goal being the best known and most put into practice Agile and DevOps. As such, following this trend of keeping itself both agile and efficient, Accenture has sought to implement more effective development methodologies in its projects thus becoming more competitive compared to other companies in the market.

To this new trend of innovating in the various technology related areas it is given, by Accenture, the name *New IT*.

Given the fact that in the context of *New IT*, and following a DevOps methodology, the automation of tasks is highly encouraged, and also the fact that there is currently not much work done in this topic in the project where this work was developed, the main focus of this work will then be the automation of test cases for a Web-based application. Throughout this report will be presented and described in detail all the work developed in this area starting with an analysis, recovery of existing and old automated tests, development of new test cases and also the integration of these on a continuous development platform.

In addition to this, there is also an analysis of possible ways and frameworks capable of evaluating the coverage of the tests on the project code providing some essential metrics for the understanding and quality analysis and rigor with which the application code is tested.

Keywords: *New IT*, DevOps, Automated Tests, Code Coverage, Automation, Continuous integration.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Devido à competição existente no mundo da consultaria, a Accenture tem vindo a apostar bastante na investigação e aplicação de novos modelos de negócio e metodologias de desenvolvimento e que possam ser aplicadas nos seus projetos a nível mundial. A esta nova tendência de inovar e reinventar os seus métodos de trabalhos é dado o nome de *New IT*.

O *New IT* é caracterizado, pela Accenture, como uma evolução mais ágil tanto da tecnologia envolvida, como da cultura empresarial, ajudando-a a desenvolver melhor, mais rápido e mais eficientemente tornando-se, assim, fundamental para a sobrevivência e nível de competitividade da empresa no mercado.[4] De modo a alcançar esses objetivos a Accenture procura aplicar estes ideais do New IT em quatro áreas principais: automatização de tarefas, analítica de dados, *multi-speed IT*, alteração organizacional.[5]

Ao longo deste capítulo serão descritos detalhadamente os pontos principais deste trabalho, desenvolvido no âmbito do *New IT*, assim como a motivação, os objetivos e uma breve contextualização da empresa e do projeto no qual o mesmo foi desenvolvido.

1.1 Motivação

Atualmente vivemos num mundo com uma crescente quantidade de software a ser desenvolvido diariamente para diversas áreas essenciais como a educação, saúde, administração. Com isto torna-se uma prioridade aumentar a rapidez e eficiência do desenvolvimento bem como testar e garantir a qualidade do mesmo de modo a mitigar os riscos envolvidos.

Realizar testes e avaliar a qualidade de um dado software nunca garante totalmente a ausência de erros no mesmo mas apenas nos mostra que estes existem e que, uma vez corrigidos, estes não deverão voltar aparecer. Isto garante uma melhoria e aumento de confiança da qualidade do software desenvolvido, tanto para a equipa de desenvolvimento como para o cliente.

Uma vez que em projetos de maiores dimensões testar manualmente o software se

pode tornar uma tarefa muito morosa e dispendiosa, automatizar a execução dos mesmos é uma realidade já posta em prática em muitas empresas. A automatização deste tipo de tarefas está muito ligada à metodologia *DevOps*. Esta deriva da junção de *Development* e *Operations* e a sua principal característica é a prática e defesa da automação e monitorização das várias tarefas relacionadas com o desenvolvimento de software.

Com a implementação do *DevOps* procura-se, principalmente, diminuir os ciclos de *deploys* e consequentemente o *time to market* bem como mitigar os potenciais erros que possam surgir nos mesmos e libertar recursos para outras tarefas críticas, melhorando a produtividade das equipas.[3] Por este motivo, em qualquer projeto, a fase respeitante à realização de testes é uma das que mais beneficia com a prática da metodologia *DevOps*, principalmente quando se tratam de projetos de maiores dimensões e com elevado número de funcionalidades tal como é o caso do projeto onde este trabalho foi realizado.

Ao contrário dos testes manuais, nos testes automáticos, deixa de ser necessário afetar uma ou mais pessoas à realização manual de testes às várias funcionalidade de um sistema podendo estas estar a realizar outras tarefas de maior relevância.

Para tal, basta apenas que seja desenvolvido um *script* para o caso de teste em questão e que possa ser executado quantas vezes for necessário. Idealmente, este tipo de *scripts* ou de automatização deverão também poder ser reutilizados ou adaptados noutros projetos ou equipas com necessidades semelhantes.

Para além disto os testes automáticos são, geralmente, bastante mais rápidos e eficientes, e é também retirado o risco da ocorrência de erro humano por parte de quem está a realizar os testes manualmente. Este fator traduz-se num aumento da confiança sobre a aplicação desenvolvida.

Recentemente foi também desenvolvida, pela *Accenture*, uma plataforma constituída por várias *frameworks open source* como: Jenkins, Bitbucket, SonarQube e Cucumber, de apoio e incentivo à prática de *DevOps*, denominada ADOP *(Accenture DevOps Platform)*. Esta plataforma foi também implementada no âmbito da iniciativa de *New It*, anteriormente referida, e procura divulgar a utilização destas ferramentas nos vários projetos da empresa, alinhando os mesmo com as novas metodologias ágeis do mercado.

1.2 Instituição de acolhimento

Este trabalho foi realizado na empresa multinacional de consultoria, Accenture. Esta empresa, antigamente conhecida por Andersen Consulting, é, atualmente, uma das maiores empresas multinacionais de consultoria e que presta os seus serviços em mais de 120 países. O prestígio e a qualidade da relação desta empresa com os seus clientes pode dever-se à forma de agir da mesma, suportada pelos seguintes valores:

- Criação de valor para o cliente
- Uma rede global

- Respeito pelo indivíduo
- As melhores pessoas
- Integridade
- Stewardship

Em Portugal a Accenture conta, atualmente, com mais de 2000 colaboradores. No total conta com mais de 400000 colaboradores que, diariamente, trabalham para mais de 40 indústrias diferentes e encontra-se dividida em cinco diferentes áreas: *Strategy*, *Consulting*, *Digital*, *Operation* e *Technology*, sendo esta última a área em que na qual este trabalho foi realizado e está associado.

1.3 Contextualização do projeto

Confidencial

1.4 Objectivos

Os principais objetivos deste trabalho foram:

1. Análise e integração de um sistema integrado de implementação e integração de casos de teste automáticos, suportada pelas *frameworks* divulgadas pela iniciativa *New IT* da Accenture.

2. Avaliação e criação de uma metodologia válida que permita às equipas funcionais do projeto especificarem e comunicarem a implementação de novos casos de teste, à equipa responsável pelos mesmos, da forma mais eficiente e sucinta possível.

3. Avaliação e integração de uma ferramenta de avaliação de cobertura servindo de complemento analítico ao testes implementados e também de incentivo à monitorização e contínuo aumento da qualidade dos testes implementados.

1.5 Contribuições

As principais contribuições destes trabalho foram então:

1. Agilização e otimização da fase de desenho e implementação de casos de teste através da criação de uma metodologia válida para as várias equipas funcionais do projeto;

2. Incrementação e melhoria do universo de casos de teste para algumas das áreas funcionais do projeto.
3. Execução contínua e automática de testes sobre as funcionalidades da aplicação Web do projeto;
4. Avaliação contínua da cobertura dos testes sobre o código da aplicação através da integração de uma *framework* de análise de cobertura de testes.

1.6 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 2 – Apresenta um breve contexto à área de realização e automatização de testes de software, descrevendo alguns conceitos relacionados com o trabalho realizado.
- Capítulo 3 – Descrição e contextualização das ferramentas utilizadas para implementação ou suporte deste trabalho.
- Capítulo 4 - Descreve a análise e contextualização da estrutura existente no projeto antes de ter sido dado início a este trabalho bem como a arquitetura, integração e utilização das várias ferramentas no mesmo.
- Capítulo 5 - Descreve toda a fase implementação, automatização e integração contínua realizada neste trabalho.
- Capítulo 6 - Apresenta os resultados obtidos após a conclusão da realização deste trabalho.
- Capitulo 7 - Apresenta uma conclusão e breve discussão dos resultados assim como uma perspetiva quanto ao possível trabalho futuro.

Capítulo 2

Trabalho relacionado

2.1 Automatização de testes

Realizar testes a um dado software desenvolvido ou em desenvolvimento é uma prática já existente há largos anos e tem vindo, ultimamente, a ganhar maior popularidade e importância uma vez que ocupa uma grande parte do tempo e esforço de desenvolvimento, entre 40-50%.[16]

Vários estudos têm vindo a mostrar que a automatização da fase de testes, nos *pipelines* de desenvolvimento de *software*, pode ter grandes impactos nos custos, tanto a nível financeiro como de recursos humanos. De acordo com um estudo analítico de *RM Sharma*[22], este concluiu que testar manualmente um sistema se pode tornar uma tarefa entediante, morosa e que requer grande custos financeiros. Por outro lado, concluiu que, apesar de ser necessário algum treino com uma dada ferramenta, de modo a ser possível automatizar testes com esta, este esforço acabará por compensar. Isto deve-se ao facto de, apesar de existir um esforço associado à implementação do *script* de teste, futuramente, deixará necessário existir intervenção humana após a implementação do mesmo, libertando esses recursos para outras tarefas.

Por este motivo, ao longo dos anos tem existido uma preocupação crescente com quais os tipos de testes que devem ser realizados e dependendo dos objetivos para tal. Sendo uma área presente nas várias etapas de desenvolvimento de software, encontra-se, assim, bem dividida em vários níveis, dependendo da escala dos sistemas a testar ou dos objetivos que se pretendem alcançar com a realização dos testes.

Dependendo dos objetivos para os quais são implementados testes, existem vários níveis diferentes e que definem aquilo que se pretende efetivamente testar. Segundo um artigo de *Shivkumar*[24], apesar de existirem bastantes níveis de testes possíveis, os principais são:

- **Unitários**: Sendo o nível mais baixo, este tem como objetivo testar cada método/módulo de um sistema e verificar o seu correto funcionamento.

- **Integração**: Tem como objetivo testar duas componentes (ex.: classes) do software e a interação entre elas de modo a garantir o correto funcionamento das mesmas após a sua integração.
- **Funcionais**: Tem como objetivo testar todos os requisitos funcionais pretendidos para um sistema.
- **Sistema**: Tem como objetivo testar o sistema como um todo tendo em atenção, principalmente, os aspetos funcionais do mesmo.
- **Aceitação**: É normalmente realizado pelo cliente do sistema após o mesmo estar concluído. Tem como objetivo verificar que o sistema desenvolvido vai de encontro ao esperado pelo cliente.
- **Beta**: É dada a oportunidade a um grupo ainda reduzido de pessoas de testar o sistema em avanço mas com o compromisso de relatar erros que possam ocorrer. Estes são bastante realizados por empresas de maior dimensão.
- **Regressão**: Testa novamente o sistema após alterações ou correções no sistema.

Para além destes níveis, de acordo com o mesmo autor, existem ainda duas metodologias diferentes que podem ser aplicadas a cada um destes anteriormente enumerados. Essas duas metodologias são então:

- **White-Box Testing**: Neste tipo de testes já existe conhecimento do design e da forma como o código do software está implementado. Isto torna possível que sejam desenvolvidos testes tendo em conta os vários caminhos possíveis percorridos ou condições existentes no código e tendo como foco a cobertura do mesmo
- **Black-Box Testing**: Nesta metodologia não existe qualquer conhecimento do padrão de desenho e implementação de código realizada no sistema. Como tal, esta é baseada puramente na funcionalidade do sistema.

Como tal, dada a elevada preocupação de assegurar a qualidade do software desenvolvido e o consequente aumento de investigação, com o objetivo de criar mais e melhores técnicas e metodologias diferentes para desenho e aplicação de testes, é de esperar que apareça também uma vasta variedade de *frameworks* e ferramentas disponíveis no mercado, para o cumprimento dos requisitos dos vários tipos de testes e agilização da implementação dos mesmos. *Mohamed Monier* e *Mahmoud Mohamed El-mahdy*[17] expõem algumas das ferramentas existentes atualmente no mercado e como estas variam entre si. Sendo algumas *open-source* e outras necessitando licença comercial para serem utilizadas, estas variam principalmente na forma como os seus *scripts* de testes são criados. Estas ferramentas permitem que, para sistemas mais dinâmicos, um programador experiente com a ferramenta programe o *script* de teste a ser reproduzido no sistema, contudo, é necessária aprendizagem e conhecimento da mesma. Por este motivo, todas estas ferramentas apresentadas permitem criar *scripts* de teste reproduzíveis gravando os movimentos de um utilizador e compilando estes, posteriormente, num caso de teste. Esta

poderá ser uma abordagem viável em sistemas Web mais estáticos e para utilizadores que não tenham ainda grandes conhecimentos de programação ou da ferramenta que se está a utilizar.

Ainda antes de existir uma Web mais dinâmica, as aplicações Web eram testadas ao nível do protocolo HTTP [7]. As equipas de teste criavam pedidos HTTP fictícios que eram enviadas à aplicação Web submetida a teste. Posteriormente, após resposta da aplicação, normalmente num formato HTML, era feita uma análise desta e determinado o sucesso da mesma. Contudo, com a rápida evolução da Web, as páginas tornaram-se também mais dinâmicas e responsivas aos utilizadores, em boa parte, graças ao aparecimento e utilização de chamadas AJAX. Por este motivo pode tornar-se preferível a programação dos próprios *scripts* de testes, permitindo que exista um maior controlo dos tempos de resposta e de *timeouts* de um sistema submetido a teste.

Por último, a abordagem tomada na realização deste trabalho é, em boa parte, semelhante à sugerida por *Christian Colombo*, *Mark Micallef* e *Mark Scerri*[8]. Estes propõem uma abordagem muito baseada em *Behaviour Driven Development* para a implementação de casos de teste. Esta abordagem tem como objetivo descrever, em primeiro lugar, o comportamento esperado para um caso de teste, através de uma linguagem quase natural, o Gherkin e, apenas posteriormente, programar o comportamento do mesmo, garantindo que são cumpridos os requisitos pedidos. Esta abordagem permite que equipas com pouco conhecimento técnico, como, por exemplo, os clientes ou equipas ligadas ao negócio, contribuam para o desenvolvimento dos testes, descrevendo em linguagem natural aquilo que pretendem que seja efetivamente testado. Para além disto estes utilizam também as mesmas ferramentas utilizadas na realização deste trabalho, Cucumber e Selenium e argumentando a favor do poder de automação de testes que é possível obter, utilizando estas ferramentas em conjunto.

2.2 Análise de cobertura

A análise ou avaliação de cobertura é, atualmente, uma técnica bastante utilizada como complemento ao desenvolvimento de testes de software. Esta permite que seja avaliado e detetado todo o código que é executado durante a realização de uma sessão de testes, ou seja, qual o código que está efetivamente a ser testado. Esta técnica é responsável por gerar algumas métricas como, por exemplo, a percentagem de linhas executadas, instruções e também funções.

A partir deste tipo de dados torna-se então mais fácil de entender e detetar que casos de teste possam eventualmente ser implementados de modo a permitir uma melhor cobertura de testes sobre um dado software. É certo que nunca se consegue garantir por completo a inexistência de faltas no software, contudo, através da análise de cobertura é possível obter melhores garantias e confiança de que um dado software se comporta da forma

prevista dado que foram realizados testes sobre o mesmo e estes correram de acordo com o esperado.

Para além disto esta técnica pode também ser aplicada em qualquer fase de testes como, por exemplo, testes unitários, integração ou de sistema, principalmente, através da instrumentação do código fonte da aplicação em causa.

2.2.1 Instrumentação de código

A instrumentação de código é uma técnica bastante utilizada de hoje em dia para permitir obter novas funcionalidades, estatísticas ou até para monitorização da execução de programas. Esta técnica funciona através da adição de novas instruções no código fonte ou já compilado, também designado *byte code*. Em tempo de execução, estas instruções adicionais são também executadas em simultâneo com o código original e realizam chamadas a bibliotecas específicas de modo a que possam ser geradas métricas de cobertura sobre o código executado. [2]

Uma vez que estas alterações são apenas aditivas, geralmente, as ferramentas que utilizam esta técnica não alteram o estado nem o comportamento do código original e, portanto, não colocam em causa o correto funcionamento do código submetido a avaliação de cobertura.

Por este motivo esta técnica é bastante utilizada atualmente para a área de testes uma vez que permite perceber quais as partes do código que são efetivamente testadas durante a execução de um dado caso de teste. Esta análise de cobertura incentiva também a que sejam desenvolvidos novos testes que exercitem o maior número possível de linhas de código de modo a obter uma maior cobertura sobre este e providenciando uma boa métrica para avaliação da qualidade do código desenvolvido.

2.2.2 Tipos de instrumentação

De acordo com a documentação de uma das ferramentas testadas ao longo deste trabalho, o JaCoCo, o processo de avaliação de cobertura de código, pode ser obtido a partir de várias abordagens diferentes. No entanto, as *frameworks* de cobertura testadas neste trabalho realizam esta avaliação recorrendo apenas à técnica de instrumentação de código Java. Na imagem apresentada abaixo pode ser observado um diagrama das várias formas de aplicação desta técnica.

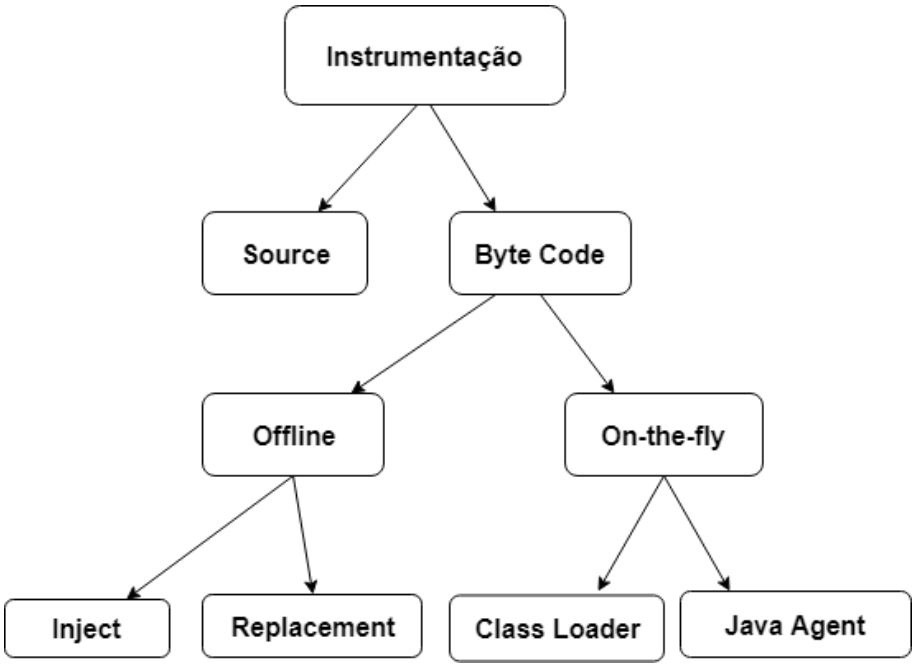

Figura 2.1: Diferentes abordagens para realização da instrumentação de código. Adaptado de [13]

Apesar da existência destas várias abordagens, o resultado final deverá ser, regra geral, o mesmo, ao nível das métricas recolhidas. Como tal e apesar disto, continuam a existir diferenças e vantagens inerentes à utilização de cada uma destas diferentes abordagens.

- ***Source*** - É adicionado novo código da biblioteca de cobertura ao código fonte, antes de este ser compilado. Para que esta funcione corretamente é necessário compilar o código fonte com a biblioteca da ferramenta de cobertura respetiva.
- ***Byte Code: Offline*** - São adicionadas instruções, em *byte code*, da biblioteca de cobertura diretamente no código compilado. Esta abordagem é mais estável dado que é assegurado que funciona em qualquer máquina virtual de Java.
- ***Byte Code: On-The-Fly*** - O código é instrumentado apenas em tempo de execução, ou seja, não há qualquer alteração direta tanto nos ficheiros do *source code* como do *byte code*. Esta abordagem é possível graças à utilização de *Java Agents* que ”intercetam”o código em tempo de execução como detalhado mais abaixo.

Esta última abordagem, *on-the-fly*, apenas passou a ser possível a partir da versão 1.5 do Java. Esta versão introduziu uma biblioteca dedicada à instrumentação de *byte code*, a *java.lang.instrument* [18], e que permitiu padronizar e facilitar a forma como esta é feita. Contudo, de modo a poder usufruir desta biblioteca é necessária a utilização de uma ferramenta adicional chamada *Java Agent*, e que, no caso do JaCoCo, é possível a sua utilização tal como será demonstrado mais a frente neste relatório.

Apesar do JaCoCo ser uma ferramenta de análise de cobertura focada para a linguagem Java, as técnicas mencionadas acima são também as utilizadas para quaisquer outras linguagens que permitam este tipo de análise.

Num estudo realizado por *Qian Yang* e *David Weiss* [27] é exposto e comparado um total de 17 diferentes ferramentas de avaliação de cobertura. Estas são utilizadas para outras linguagens de programação como, por exemplo, FORTRAN, .net, C#, C++/C, PHP e COBOL. Apesar das grandes diferenças entre estas linguagens, as técnicas utilizadas para a instrumentação destas são as mesmas expostas anteriormente, *source code*, *byte code* ou *on-the-fly*. Este estudo expõe também as várias métricas avaliadas por cada uma das 17 ferramentas assim como o tipo de relatórios gerados por estas.

Deste modo, tendo em conta a variedade de ferramentas existentes atualmente e a semelhança do modo de operação das mesmas, cabe às equipas de desenvolvimento e de testes escolher aquelas que melhor se adequem aos seus projetos tendo em conta as suas necessidades e a eficiência das mesmas com base neste tipo de estudos.

Capítulo 3

Ferramentas utilizadas

Embora atualmente exista um grande leque de ferramentas disponíveis para apoio à pratica de *DevOps*, a escolha daquelas utilizadas neste trabalho deveu-se principalmente ao facto de serem também as utilizadas pela Accenture a nível global.

Com a exceção das ferramentas de análise de cobertura, as outras principais ferramentas utilizadas, Selenium , Cucumber e Jenkins são as utilizadas pela Accenture na sua plataforma ADOP, para apoio à prática de *DevOps*. Embora esta plataforma não tenha sido diretamente utilizada neste trabalho, por questões de acesso à rede do cliente a partir da mesma, foram, de qualquer forma, utilizadas as ferramentas disponibilizadas nesta plataforma por questões de concordância com as políticas e boas práticas da empresa.

Ao longo deste capítulo será feita uma breve contextualização e descrição da utilidade das várias ferramentas utilizadas na fase de implementação deste trabalho.

3.1 Linguagem de programação

O Java foi a linguagem escolhida para a fase de desenvolvimento deste trabalho. Esta é uma linguagem de programação orientada a objetos [19] e foi, de acordo com *ranking* da *IEEE Spectrum*, a terceira linguagem mais utilizada mundialmente para desenvolvimento de software no ano de 2017 [23]. Esta foi desenvolvida inicialmente, pela empresa Sun Microsystems, que foi mais tarde adquirida pela empresa Oracle.

A linguagem Java apareceu pela primeira vez em 1995 e foi desenvolvida com o objetivo de ser também bastante flexível permitindo que um qualquer programa pudesse ser executado independentemente do sistema ou máquina em que foi desenvolvido. Para tal bastava apenas que a máquina de destino conseguisse compreender o código compilado, através da instalação do Java Virtual Machine (JVM). Por este motivo o seu *slogan* era a sigla *WORA* que significa *Write Once, Run Anywhere*.

Como tal, tornou-se rapidamente numa das linguagens mais utilizadas mundialmente para qualquer tipo de desenvolvimento.

Dado que o Java é suportado pelas ferramentas utilizadas neste trabalho principal-

mente o Selenium, e é também a linguagem utilizada no desenvolvimento do projeto onde este trabalho foi integrado, esta foi, na versão 1.8, a linguagem escolhida e utilizada para a implementação e automatização dos testes.

3.2 Apache Maven

O Apache Maven é uma ferramenta de automatização da compilação de projetos, principalmente desenvolvidos em Java.[6] Esta ferramenta tornou-se muito popular uma vez que permite realizar a devida configuração de todo o processo de compilação de um projeto bem como as dependências necessárias ao mesmo, através de um único ponto central de informação.

Este ponto central é um ficheiro de extensão XML denominado POM (*Project Object Model*). Neste ficheiro XML pode ser descrito a ordem de compilação de ficheiros, pastas com informação necessária para tal ou a indicação de dependências de módulos ou *plugins* externos. Estas dependências são automaticamente descarregadas, a partir do repositório remoto da própria ferramenta, e armazenadas no repositório local da máquina onde serão utilizadas.

3.3 Git

O Git é um sistema, *open-source*, de controlo de versões de ficheiros e desenvolvido para ser eficiente tanto em pequenos como em grandes projetos.[12] Este sistema permite, com facilidade gerir o código fontes das aplicações em desenvolvimento, guardando versões anteriores destes ficheiros e permitindo atualizar, de forma eficaz ficheiros sobre os quais tenham sido feitas alterações recentes.

Frequentemente pode acontecer que, em projeto de maiores dimensões, exista mais do que uma pessoa a efetuar uma alteração numa dada parte do código. Posteriormente pode também acontecer que, ao tentar guardar estas novas alterações remotamente, possam existir conflitos com novas versões criadas por outra pessoa. Nestes casos, o Git permite também de forma fácil resolver estes conflitos e decidir qual das versões do código é a mais recente e aquela que deve ser guardada e mantida como a mais atualizada.

Não existindo necessidade de trabalhar em equipa, o Git pode também ser utilizado para o desenvolvimento de código localmente, não necessitando de ligação à rede ou um servidor de hospedagem do código. Embora existam vários servidores para este fim, neste trabalho foi utilizado o servidor escolhido pela Accenture, para hospedagem dos seus projetos, o BitBucket.

Para além disto, o Git pode também ser utilizado via terminal ou através de um IDE que permita a sua integração. Durante a realização deste trabalho, o Git foi utilizado sob a forma de interface gráfica, integrada no IDE Eclipse uma vez que este permite a instalação

do *plugin* necessário para tal.

3.4 Eclipse IDE for Java EE Developers

Para o desenvolvimento de grande parte deste trabalho foi utilizado o IDE (*Integrated Development Environment*) Eclipse [11]. Este é uma das mais utilizadas ferramentas para desenvolvimento de software devido, em boa parte, ao seu suporte a várias linguagens de programação. Este permite também um alto nível de customização uma vez que suporta um elevado número de *plugins* desenvolvidos pela sua comunidade de utilizadores com o objetivo de fornecer excelentes ferramentas adicionais de apoio ao desenvolvimento, integradas no próprio IDE.

Como tal, o Eclipse permitiu integrar, via Maven, e utilizar de forma intuitiva e eficaz, as principais *frameworks* utilizadas neste trabalho, o Selenium e o Cucumber assim como toda a lógica do repositório remoto do código, via Git.

Sendo também o IDE utilizado para o desenvolvimento do projeto onde este trabalho foi integrado, a utilização do mesmo tornou-se assim uma escolha óbvia para o desenvolvimento deste trabalho.

Apesar de suportar várias linguagens de programação, o desenvolvimento deste IDE foi feito, maioritariamente em Java e, como tal, a sua principal utilização e suporte é para todo o tipo de desenvolvimento feito nesta linguagem.

3.5 Cucumber

Trata-se de uma *framework open-source*[9] de apoio ao desenvolvimento de testes de aceitação e baseada numa metodologia de desenvolvimento *Behaviour-Driven Development* (BDD) [10].

Esta metodologia nasceu no âmbito das técnicas de desenvolvimento *Agile* e procura, principalmente, promover uma boa comunicação entre as equipas de desenvolvimento e as equipas diretamente envolvidas no negócio. Este foco na comunicação entre ambas as equipas tem como principal objetivo garantir que o software desenvolvido vai de encontro às expectativas do cliente.

O ciclo típico do desenvolvimento de software apoiado nesta metodologia caracteriza-se por ter, em primeiro lugar, as equipas de desenvolvimento e de negócio a comunicar e desenhar, em conjunto, alguns cenários de utilização do sistema. Após acordo entre ambas as partes sobre o desenho dos cenários, é dado início ao desenvolvimento de software com o objetivo de satisfazer o correto funcionamento dos cenários criados, realizando testes sobre os mesmos. Deste modo existe uma maior garantia que o software desenvolvido cumpre com o que foi proposto e pedido pela equipa de negócio.

Uma vez que numa metodologia BDD os requisitos e comportamento das funcionalidades não são escritas apenas pelas equipa de desenvolvimento, existe uma necessidade de que estes sejam especificados numa linguagem que possa ser entendida por todas as equipas envolvidas no desenho dos cenários de modo a facilitar a comunicação, discussão e desenho destes. [16] Para esse efeito, o Cucumber recorre à linguagem Gherkin que é utilizada pelo seu *parser* interno permitindo a associação de cada passo de um caso de teste ao código respetivo.

Assim, o objetivo principal do Cucumber é permitir que a equipa de desenvolvimento e a de negócio desenvolvam, em conjunto, cenários de utilização numa linguagem percetível por ambos e permitir que esses cenários sejam associados a bibliotecas de automação que, posteriormente, verificam o cumprimento com sucesso do cenário desenvolvido.

O uso de uma metodologia BDD, apoiada pelo Cucumber, para a automatização de cenários de teste permite também obter uma documentação atualizada e bastante explícita sobre estes e facilitar a passagem de informação sobre novos cenários a desenvolver entre as equipas de desenvolvimento e a de testes. Para além disto, o Cucumber é também o responsável por desencadear a execução dos cenários de teste pretendidos através da identificação das *tags* escolhidas e a associação das mesmas ao código da implementação dos cenários, feito através da *framework* Selenium.

3.5.1 Gherkin

Dado que, por muitas vezes, se pode tornar difícil, devido a falhas de comunicação, identificar e especificar todos os requisitos pretendidos para um projeto torna-se necessário utilizar uma linguagem que seja facilmente entendida por todos os elementos envolvidos nesta fase. Para esse efeito, numa metodologia BDD é, por muitas vezes, utilizada a linguagem Gherkin. Esta linguagem é caracterizada por ter uma sintaxe semelhante à linguagem natural humana, ter como prioridade a legibilidade por humanos e ser independente da linguagem de programação utilizada para desenvolvimento dos projetos e respetivos testes. Apesar de poder ser escrita em qualquer linguagem humana, o Gherkin segue sempre uma estrutura bem definida para descrever os casos de teste. Embora existam mais, algumas das principais palavras chave usadas neste trabalho para definir essa estrutura são [26]:

- ***Feature*:** Breve descrição do ficheiro que engloba os vários casos de teste referentes a uma mesma funcionalidade.
- ***Scenario*:** Breve descrição do caso de teste.
- ***Given*:** Descrição das pré-condições e estado inicial do caso de teste.
- ***When*:** Descrição das ações tomadas pelo utilizador durante o caso de teste.
- ***Then*:** Descrição do resultado das ações tomadas na cláusula When.

- ***@tags*:** Identificadores utilizados para indicar à classe *Runner* do Cucumber qual o caso de teste a correr.
- **"Argumentos":** As palavras entre aspas e a verde são argumentos passados ao método Java respetivo.

Além de servir como complemento da documentação o Cucumber serve como uma ferramenta de controlo de fluxo de execução dos testes uma vez que cada passo de cada caso de teste escrito em Gherkin está associado um método escrito numa linguagem de programação, neste caso, em Java.

3.6 Cucumber Extent Report

Trata-se de uma ferramenta, também *open-source*[1], para integração e complemento ao Cucumber sendo a partir desta que é gerado o relatório da execução dos testes automáticos, num formato sucinto e legível, e que contempla as principais estatísticas da execução dos mesmos.

Os relatórios gerados a partir desta ferramenta são o resultado final da execução de testes e que devem ser informados e analisados tanto pelas equipas de testes como pelas equipas de desenvolvimento, tornando-se uma mais valia para estas. A partir destes relatórios é possível perceber, em caso de falha, qual a ação que estava a ser executada e qual o erro causador da mesma. A API desta ferramenta permite também que sejam também capturados e anexados *screenshots* da página *Web* corrente. Este fator é também bastante relevante uma vez que estes *screenshots* permitem mais facilmente, em caso de erro, identificar e perceber a causa desse erro, fornecendo uma contextualização do estado da aplicação no momento da ocorrência deste.

Algumas das estatísticas recolhidas e contempladas, por esta ferramenta, no relatório gerado são, por exemplo, o tempo de execução de cada caso de testes, a percentagem que foi executada com sucesso e com erro, o tempo total de execução assim como alguma informação sobre a máquina onde foram executados os testes. Para além disto, em caso de erro, apresenta também a exceção Java que foi lançada na ocorrência do mesmo.

Sempre que é executada uma sessão de testes é gerado, pelo Cucumber, um ficheiro em formato Json e que contempla os dados principais de execução anteriormente mencionados. Dado que, com grandes quantidades de dados, um ficheiro em formato Json se torna pouco legível para humanos esta ferramenta torna possível a conversão dos mesmos para um formato mais legível e esteticamente apelativo.

3.7 Selenium

Uma grande parte das aplicações desenvolvidas de hoje em dia são aplicações Web ou Web-based sendo, assim, necessária a utilização de um *browser* para as conseguir

executar e testar. Com o crescente desenvolvimento de aplicações Web, tornou-se assim necessário arranjar uma forma de as conseguir testar de forma automatizada, realizando testes sobre as interfaces dos *browsers*. Como tal, o Selenium nasceu desta necessidade de automatizar testes para aplicações Web através da interações com os *browsers*.

O Selenium[21] é uma *framework open-source* [20] e o seu objectivo é, principalmente, permitir a manipulação e simulação da interação de um utilizador com um *browser* enviando-lhe as instruções que compõem os vários passos de um caso de teste. Esta é também uma das *frameworks* mais utilizadas atualmente para este fim não só devido ao seu suporte a várias linguagens de programação como Java, Perl, C#, Python, como também ao suporte aos vários sistemas operativos, Windows, Mac, Linux e aos vários *browsers* mais conhecidos, Chrome, Firefox, Internet Explorer, Opera e Safari.[14] Esta apresenta também uma API muito bem padronizada para a programação dos *scripts* de teste e posterior comunicação com os respetivos *browsers.*

Por este motivo esta tornou-se também a *framework* escolhida pela Accenture a nível global, para a automatização de tarefas, e principalmente, a automatização de testes em aplicações Web.

3.7.1 Selenium WebDriver

De modo a que a criação dos *scripts* automáticos, utilizando o Selenium, seja feita de forma padronizada, intuitiva e possa ser comunicada ao *browser* são utilizadas ferramentas denominadas *Webdrivers* [25]. Para cada *browser* suportado pelo Selenium existe uma *driver* respetiva. Estas são, na sua maioria, desenvolvidas pela comunidade de utilizadores e suportadas pelas equipas de desenvolvimento dos próprios *browsers*. Estas *drivers* são, basicamente, ficheiros executáveis e os responsáveis por comunicar ao *browser* respetivo os comandos programados nos *scripts* em Selenium.

Estas funcionam implementando um servidor HTTP que permita receber estes comandos a executar no *browser*, no endereço local da máquina, 127.0.0.1. Para que este servidor seja ativado e seja aberta uma nova instância, ou janela, do *browser* pronta a receber as instruções é necessário proceder à sua instanciação através do Java. Esta instanciação define qual o ficheiro executável da *driver* que será utilizado assim como permite que sejam também definidas algumas opções adicionais que melhorem a estabilidade ou a resposta do *script* a comportamentos inesperados durante a execução dos testes. O comando enumerado de seguida é o utilizado para proceder à instanciação de uma nova sessão do *browser* Internet Explorer, no Java, e é executado para cada caso de teste.

- ***WebDriver driver = new InternetExplorerDriver();***

Durante a instanciação da nova sessão do *browser* é então criado o objeto *driver* que é o objeto mais importante e transversal a todos os casos de teste dado que é sobre este que são feitas as chamadas de funções que definem os comportamentos pretendidos ao longo

do *script* de teste. Para além da criação do objeto *driver* é também aberta uma janela do *browser* sobre a qual são realizadas as interações implementadas com o Selenium.

Após a instanciação da *driver*, o servidor HTTP desta nova instância fica pronto para receber os comandos programados com a API do Selenium e que são convertidos para pedidos HTTP. Após a receção de um pedido HTTP por parte do servidor, este é transformado em comandos que possam ser executados no *browser*. Na sua maioria, a execução destes comandos é feita recorrendo aos motores de JavaScript embutidos na grande maioria dos *browsers* utilizados atualmente.

Alguns exemplos de comandos programados em Java e utilizando a API do Selenium podem ser observados na imagem 3.1 apresentada abaixo. Nesta é possível verificar alguns comandos da API do Selenium como: *get*, que navega no *browser* para o endereço dado, o *findElement*, que procura elementos HTML na página atual do *browser* e, neste caso ainda, o *sendKeys* que escreve num elemento HTML que o permita como, por exemplo, caixas de texto, a informação dada. Neste caso escreve as *Strings user* e *pass* que são passadas como argumentos do método a partir do passo Gherkin associado. Estes métodos referidos anteriormente são chamados/executados sobre o objecto *driver* da API do Selenium permitindo que estes comandos sejam então realizados no *browser*, simulando a interação com o mesmo.

```
@Given("^Fazer login Qualidade com \"([^\"]*)\" e \"([^\"]*)\"$")
public void fazer_login_Qualidade(String user, String pass) throws Throwable {
    driver.get("http://10.141.3.41:6500/spl/cis.jsp");
    driver.findElement(By.id("userId")).sendKeys(user);
    driver.findElement(By.id("password")).sendKeys(pass);
    scenario.write("A fazer login em Qualidade com as seguintes credenciais User: "
    + user + " e Pass: " + pass + "\n");

    screenshot();
```

Figura 3.1: Função Java respetiva ao passo de Login, no ambiente de Qualidade e utilizando a API do Selenium

3.8 Jenkins

Como mencionado anteriormente, o *DevOps* é uma prática muito utilizada de hoje em dia defendendo a automatização das várias tarefas ligadas ao desenvolvimento e procurando melhorar a produtividade e eficiência das equipas durante todo este processo. Como tal, ao longo dos anos têm também surgido as mais variadas ferramentas para o apoio a esta prática.

Uma das ferramentas mais conhecidas e utilizadas atualmente para apoio à prática de *DevOps* é o Jenkins [15]. Este é caracterizado por ser *open-source* [14] e que funciona como um servidor de automatização permitindo uma integração e entrega contínua através

da criação de *pipelines* de automatização das várias tarefas que compõem os processos de *build* e *deploys* de um ou mais projetos.

Para além disto o Jenkins é também bastante flexível e robusto permitindo, atualmente, a instalação e integração de mais de 1400 *plugins* de apoio à automatização das várias tarefas ligadas ao desenvolvimento e desenvolvidos, na maioria, pela sua comunidade de utilizadores. Neste trabalho foram instalados e utilizados alguns destes plugins adicionais como:

- **Email Extension** - Permite a criação e envio dos emails dos relatórios dos testes automáticos que são enviados às equipas diariamente;
- **Maven Integration** - Permite a integração de projetos estruturados em Maven no Jenkins;
- **Timestamper** - Permite adicionar um Timestamp a todas as mensagens criadas durante a execução de um Job, complementando a informação nos *logs*.

Assim, o Jenkins funciona permitindo a criação de várias tarefas, denominadas por *Jobs*, altamente configuráveis através de *scripts* ou *plugins* e com o objetivo de parametrizar o comportamento desejado em cada uma dessas tarefas. As tarefas podem ser configuradas para as seguintes etapas/opções:

- **Source Code Management** - Configurações ao nível do repositório onde se encontra o código a executar;
- **Build Triggers** - Calendarização da execução da tarefa;
- **Build Environment** - Definir opções relativas ao ambiente da tarefa como, por exemplo, *timeouts* caso esta se encontra parada, limpar as pastas antes de cada *build*, ativar/desativar *timestamps* da execução...
- **Pre-Build** - Execução de *scripts* que preparem o ambiente para o correto funcionamento de uma tarefa. Podem ser, por exemplo, *shell scripts*, *Windows batchs*, partes de uma configuração Maven, etc...
- **Build** - Definir parâmetros/argumentos a passar ao programa da tarefa ou, no caso de um projeto Maven definir o ficheiro de estruturação da *build*, pom.xml.
- **Post-Build** - Permite novamente a execução de *scripts* mas após o processo de *build* da tarefa. Neste trabalho, usando o *plugin* referido anteriormente, é nesta etapa que é feita a construção e envio do email com o relatório de execução dos testes.

Deste modo o Jenkins permite automatizar com um elevado nível de controlo as várias fases de configuração, compilação e execução de vários projetos sem qualquer intervenção humana após a sua configuração inicial. Isto permite poupar recursos financeiros ao mesmo tempo que reduz o risco de existência de erro humano em qualquer das fases envolvidas.

3.9 Análise de Cobertura

Durante a realização deste trabalho foram avaliadas algumas das ferramentas mais utilizadas atualmente para avaliação de cobertura de código Java. Esta análise foi realizada com o objetivo de escolher, de entre as opções, aquela que melhor se adequaria à integração no projeto em questão e servisse como um complemento analítico à execução do testes automáticos.

A análise e comparação destas *frameworks* foi realizada com o intuito de conseguir avaliar quais as partes do código que eram efetivamente testadas para os casos de teste já implementados. Para além disto pretendeu-se também incentivar à melhoria da qualidade de desenho de casos de testes futuros tendo por base, os relatórios de cobertura gerados juntamente com cada execução diária dos testes.

As *frameworks* tidas em conta foram então as seguintes:

- **Cobertura**
- **JaCoCo (Java Code Coverage)**
- **EclEmma**

Estas três ferramentas funcionam de modo semelhante permitindo realizar a instrumentação de código de modo a conseguir detetar quais as partes deste que são executadas durante um dado cenário de testes. A partir da execução do código instrumentado são gerados dados estatísticos, escritos num ficheiro de formato inerente à ferramenta utilizada, que contemplam informação sobre a execução dos mesmos. Posteriormente, a partir destes dados, cada uma destas ferramentas permite que seja gerado um relatório num formato legível, semelhante a uma página Web (HTML), e contemplando a informação estatística recolhida de modo a poder ser analisada e tida em conta na criação de novos casos de teste ou melhoria dos já existentes.

Um dos principais fatores nas escolha destas *frameworks* foi devido ao facto de as três serem *open-source*. Dado que este trabalho se inseriu no âmbito do *New IT*, e também dada a sua clara prioridade à utilização de ferramentas *open source*, este foi o factor principal para a escolha das mesmas.

No capítulo 4 pode ser observada a forma como estas três ferramentas foram avaliadas e qual destas foi a escolhida para ser integrada e utilizador no projeto servindo de complemento estatístico aos testes automáticos implementados.

Capítulo 4

Análise

Confidencial

Capítulo 5

Implementação

Confidencial

Capítulo 6

Resultados

Confidencial

Capítulo 7

Conclusão

Confidencial

Bibliografia

[1] Código Fonte Cucumber Extent Reporter. https://github.com/email2vimalraj/CucumberExtentReporter. Último acesso 7 Dezembro 2017.

[2] Jari Aarniala. Instrumenting java bytecode. In *Seminar work for the Compilers-course, Department of Computer Science, University of Helsinki, Finland*, 2005.

[3] Accenture. Building new IT agility. https://www.accenture.com/us-en/blogs/blogs-building-new-it-agility-going-beyond-devops-team, 2017. Último acesso 12 Dezembro 2017.

[4] Accenture. New IT - Unleashing the new. https://www.accenture.com/us-en/insight-new-it-power-digital-curve, 2017. Último acesso 12 Dezembro 2017.

[5] Accenture. New IT - Your business imperative. https://www.accenture.com/us-en/insight-rotating-new-it-no-longer-option, 2017. Último acesso 12 Dezembro 2017.

[6] Apache. Apache Maven Website. https://maven.apache.org/. Último acesso 8 Julho 2018.

[7] Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. Web application tests with selenium. *IEEE software*, 26(5), 2009.

[8] Christian Colombo, Mark Micallef, and Mark Scerri. Verifying web applications: from business level specifications to automated model-based testing. *arXiv preprint arXiv:1403.7258*, 2014.

[9] Cucumber. Código Fonte Cucumber. https://github.com/cucumber/cucumber-jvm. Último acesso 7 Dezembro 2017.

[10] Cucumber. Cucumber Website. https://cucumber.io/. Último acesso 7 Dezembro 2017.

[11] Eclipse Foundation. Eclipse Website. `https://www.eclipse.org/ide/`. Último acesso 24 Maio 2018.

[12] Git. Git Website. `https://git-scm.com/`. Último acesso 8 Julho 2018.

[13] JaCoCo. JaCoCo Documentation. `https://www.jacoco.org/jacoco/trunk/doc/implementation.html`. Último acesso 6 Junho 2018.

[14] Jenkins. Código Fonte Jenkins. `https://github.com/jenkinsci/jenkins`. Último acesso 12 Dezembro 2017.

[15] Jenkins. Jenkins Website. `https://jenkins-ci.org`. Último acesso 12 Dezembro 2017.

[16] Lu Luo. Software testing techniques. *Institute for software research international Carnegie mellon university Pittsburgh, PA*, 15232(1-19):19, 2001.

[17] Mohamed Monier and Mahmoud Mohamed El-mahdy. Evaluation of automated web testing tools. *International Journal of Computer Applications Technology and Research*, 4(5):405–408, 2015.

[18] Oracle. Java Instrumentation Documentation. `https://docs.oracle.com/javase/7/docs/api/java/lang/instrument/package-summary.html`. Último acesso 8 Junho 2018.

[19] Oracle. Java Website. `https://www.oracle.com/java/index.html`. Último acesso 24 Maio 2018.

[20] SeleniumHQ. Código Fonte Selenium . `https://github.com/SeleniumHQ/selenium`. Último acesso 7 Dezembro 2017.

[21] SeleniumHQ. Selenium Website. `http://www.seleniumhq.org/`. Último acesso 7 Dezembro 2017.

[22] RM Sharma. Quantitative analysis of automation and manual testing. *development*, 4(1), 2014.

[23] IEEE Spectrum. Ranking do Top de linguagens utilizadas em 2017. `https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages`. Último acesso 24 Maio 2018.

[24] Shivkumar Hasmukhrai Trivedi. Software testing techniques. *International Journal of Advanced Research in computer science and software Engineering*, 2(10), 2012.

[25] Fei Wang and Wencai Du. A test automation framework based on web. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 683–687. IEEE, 2012.

[26] Matt Wynne and Aslak Hellesoy. *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2012.

[27] Qian Yang, J Jenny Li, and David M Weiss. A survey of coverage-based testing tools. *The Computer Journal*, 52(5):589–597, 2009.

Capítulo 8

Anexos

Confidencial